10. Recursividad y ordenación

Programación - 1º DAM

Luis del Moral Martínez

versión 20.10

Contenidos del tema

1. Recursividad
2. Ordenación

1. Recursividad

Concepto de recursividad (1)

- Una **función recursiva** es aquella que se llama a sí mismo
- Esto permite abordar el problema desde el prisma de la **recursividad**
- **Algunos ejemplos de uso:**
  - Ordenación
  - Cálculos matemáticos (factorial, Fibonacci…)
  - Resolución de problemas (Torres de Hanoi…)

1. Recursividad

Concepto de recursividad (2)

- Para que una función recursiva sea correcta:
  - No se debe generar una secuencia infinita de llamadas (bucle infinito)
  - Debe incluir un componente base (es la condición de salida para evitar el bucle infinito)
- **Ejemplo (función factorial)**:

$$f(n)\begin{cases}1 & n = 1 \\ n \cdot f(n-1) & n > 1\end{cases}$$

La **condición de salida** o base es f(n) = 1

1. Recursividad

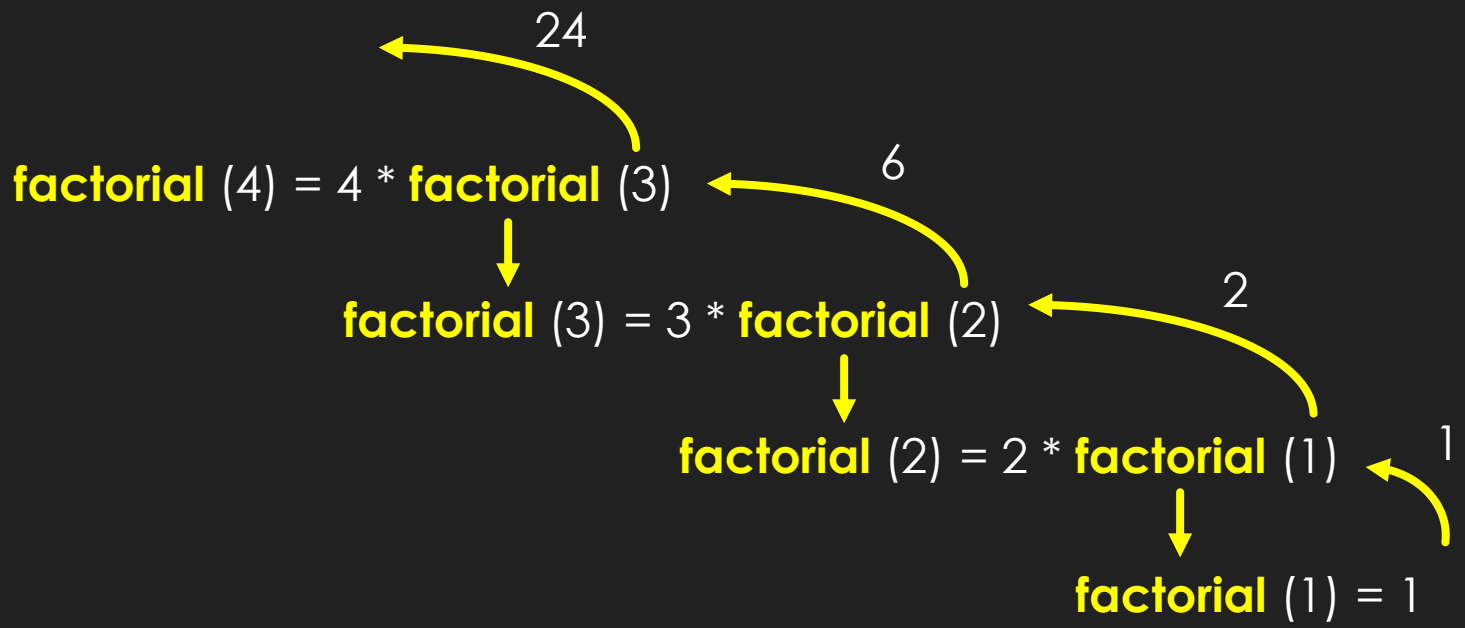
Recursión vs. iteración

- Las soluciones iterativas utilizan estructuras repetitivas
- Las soluciones recursivas utilizan, principalmente, una estructura de selección
- La recursión tiene ciertas **desventajas**:
  1. Incrementa el tiempo de llamada a las funciones y el tamaño de la pila
  2. Mayor consumo de recursos
- Sólo se debería emplear en aquellos problemas que tengan **naturaleza recursiva**
- Abre el fichero de ejemplo **10_01_recursividad.cpp**

1. Recursividad

Ejemplo factorial recursivo (paso a paso)

- A continuación se explica el cálculo del factorial recursivo



2. Ordenación

Concepto de ordenación

- La **ordenación** permitirá almacenar los datos ordenados según un criterio
- La ordenación puede efectuarse en un vector, en una lista, en ficheros…
- A continuación analizaremos algunos de los algoritmos de ordenación más conocidos:
  - Ordenación por **intercambio**
  - Ordenación por **burbuja**
  - **Quicksort** (ordenación rápida)

2. Ordenación

Ordenación por intercambio (1)

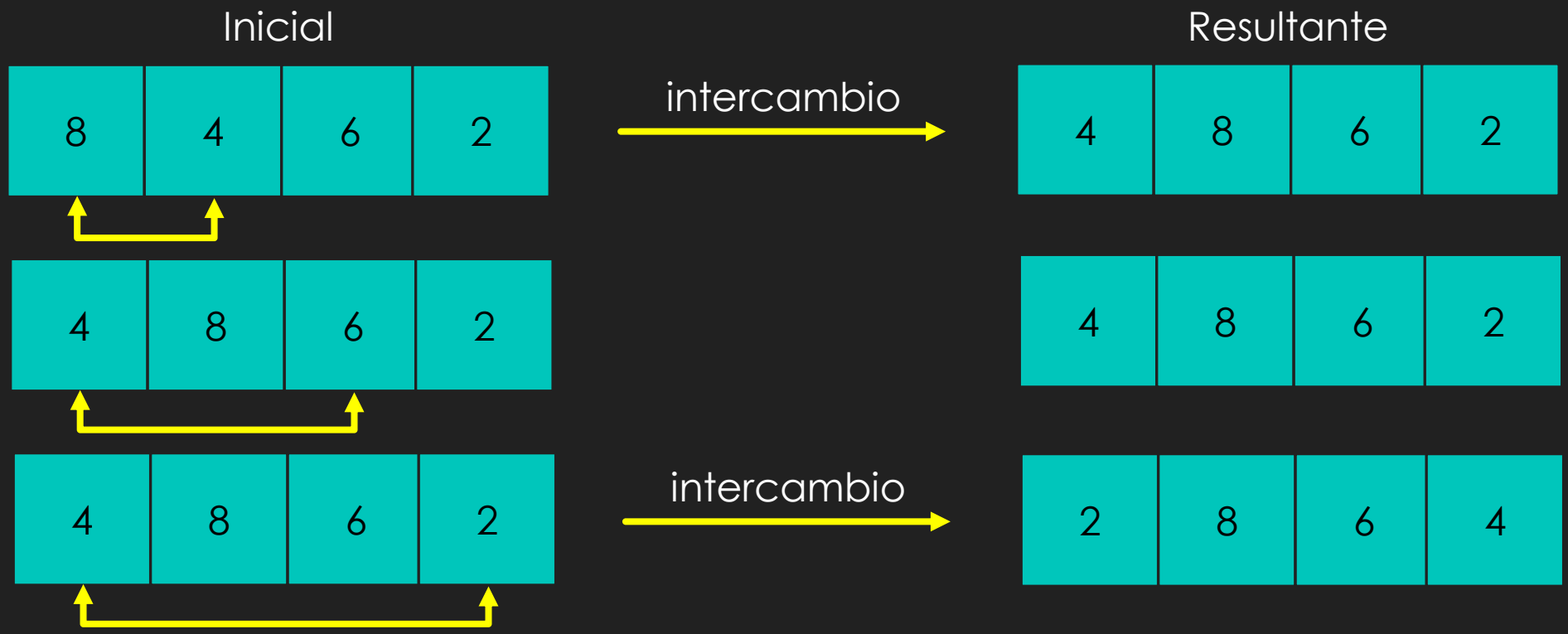
- Es el algoritmo más sencillo
- Ordena los elementos en una lista en orden ascendente
- Se realizan comparaciones del elemento inferior con los restantes
- El algoritmo realiza **n – 1** pasadas, donde n es el número de elementos
- **Ejemplo**: sea la lista 8 4 6 2

8	4	6	2

2. Ordenación

Ordenación por intercambio (2)

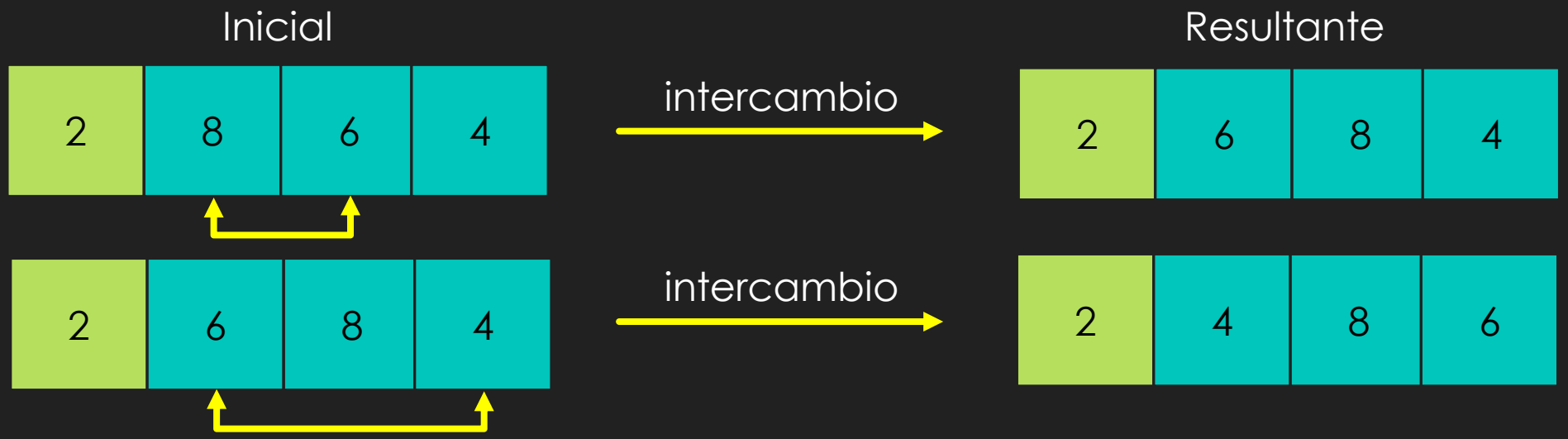
Pasada 1



2. Ordenación

Ordenación por intercambio (3)

Pasada 2



2. Ordenación

Ordenación por intercambio (4)

Pasada 3

2. Ordenación

Ordenación por burbuja (1)

- Compara elementos adyacentes
- El algoritmo realiza **n – 1** pasadas, donde n es el número de elementos
- **Ejemplo**: sea la lista 25 60 45 35 12 92 85 30

25	60	45	35	12	92	85	30

2. Ordenación

Ordenación por burbuja (2)

Pasada 1

25	60	45	35	12	92	85	30	
25	60	45	35	12	92	85	30	intercambio
25	45	60	35	12	92	85	30	intercambio
25	45	35	60	12	92	85	30	intercambio
25	45	35	12	60	92	85	30	
25	45	35	12	60	92	85	30	intercambio
25	45	35	60	12	85	92	30	intercambio

2. Ordenación

Ordenación por burbuja (3)

Pasada 1	25	45	35	12	60	85	30	92
Pasada 2	25	35	12	45	60	30	85	92
Pasada 3	25	12	35	45	30	60	85	92
Pasada 4	12	25	35	30	45	60	85	92
Pasada 5	12	25	30	35	45	60	85	92
Pasada 6	12	25	30	35	45	60	85	92
Pasada 7	12	25	30	35	45	60	85	92

Las dos últimas pasadas no son necesarias (ineficacia)

2. Ordenación

Quicksort (1)

- Fue creado por **Tony Hoare**
- Se basa en la **división** en particiones de la lista que vamos a ordenar
- Se trata del algoritmo más eficiente, elegante y pequeño
- La lista se divide en tres partes: **segmento izquierdo**, **pivote** y **segmento derecho**
- El algoritmo utiliza la **recursividad** para ejecutar Quicksort sobre cada parte de la lista
- La elección del pivote es importante (la ordenación depende de este elemento)
- **Ejemplo**: sea la lista 5 2 1 9 3 8 7

2. Ordenación

Quicksort (2)

- **Vamos a considerar que el pivote es el primer elemento de la lista**
- **Existen diferentes implementaciones del algoritmo**

pivote = 5

izquierda-1 (elementos menores que el pivote)

derecha-1 (elementos menores que el pivote)

2. Ordenación

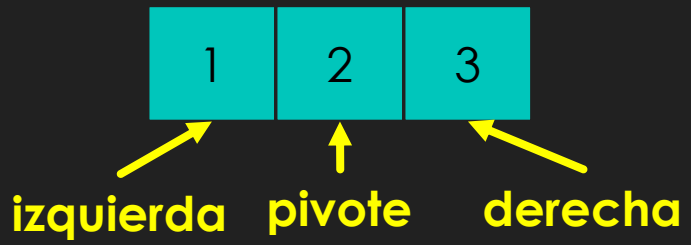
Quicksort (3)

izquierda-1 (elementos menores que el pivote)

pivote = 2

izquierda (elementos menores que el pivote)

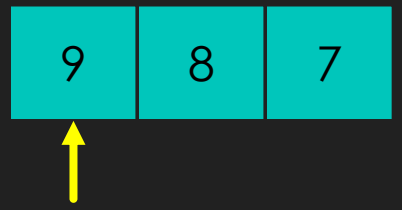
derecha (elementos mayores que el pivote)



2. Ordenación

Quicksort (3)

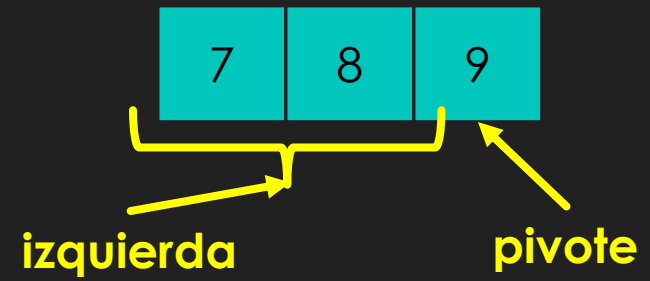
derecha-1 (elementos mayores que el pivote)



pivote = 9

izquierda (elementos menores que el pivote)

derecha (elementos mayores que el pivote)



2. Ordenación

Quicksort (4)

- **Lista ordenada final**

1	2	3	5	7	8	9

Créditos de las imágenes y figuras

Cliparts e iconos

- **Obtenidos mediante la herramienta web IconFinder** (según sus disposiciones):
  - Diapositiva 1
  - Según la plataforma IconFinder, dicho material puede usarse libremente (free comercial use)
  - A fecha de edición de este material, todos los cliparts son free for comercial use (sin restricciones)

Resto de diagramas y gráficas

- Se han desarrollado en PowerPoint y se han incrustado en esta presentación
- Todos estos materiales se han desarrollado por el autor
  - Si se ha empleado algún icono externo, este se rige según lo expresado anteriormente